Services webs et protocoles associés

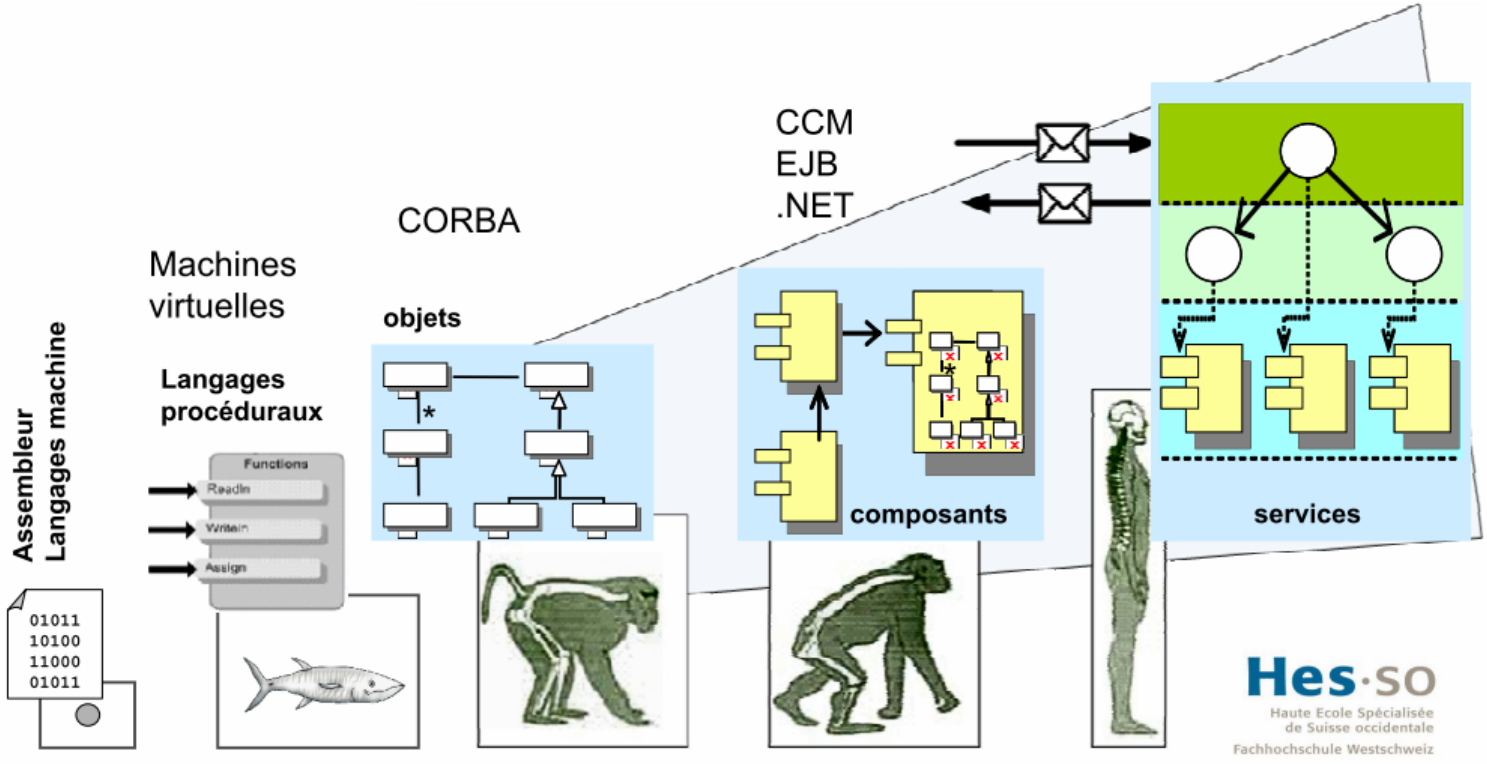
M. PERREIRA DA SILVA

Version PDF des slides

Introduction

- Oublions le web quelques instants...
- Parlons programmation et architecture logicielle
  - Ne **pas réinventer la roue** à chaque programme
  - **"Industrialiser"** le développement (ex: sous traitance)
    - Énormes logiciels d'entreprise
  - Besoin de **réutilisation**
    - Du code : Programmation orientée objet (POO)
    - Des fonctionnalités : **Programmation orientée composant** (POC)

Évolution du niveau d'abstraction



Composant ?

- Entité **indépendante**, fournissant une **fonctionnalité**
  - **Interface** de communication prédéfinie
  - Documentation séparée
  - Tests séparés
- **Programmation orientée composant** = assemblage de différents composants
  - Différentes provenances
  - Différents langages (objets ou non)

Composants et architecture "locale"

- Les composants résident **sur une même machine**
- Besoin d'une plateforme de gestion de composants
  - Component Object Model (COM): OCX, ActiveX (Microsoft)
  - XPCOM: modèle de composant de Mozilla
  - Etc.
- Définition de l'**interface du composant** via un langage
  - **IDL:** Interface Description Langage (équivalent du .h en C)
  - Gestion de l'hétérogénéité des types dans les différents langages

Composants et architectures "distribués"

- Architecture **distribuée**
  - Les composants peuvent être sur différents ordinateurs
  - Appels de méthode / procédures distants
- Contraintes complémentaires
  - Besoin d'un **protocole de communication** entre les composants
  - Besoin de **sérialiser / désérialiser** les données sur le réseau
    - Binaire
    - Texte
  - Besoin de **détection et description** des composants sur le réseau
- Exemples:
  - DCOM: version distribuée de COM (Microsoft)
  - CORBA: standard industriel (OMG)
  - .Net Remoting
  - Java Entreprise Edition (JEE) et les Entreprise Java Bean (EJB)

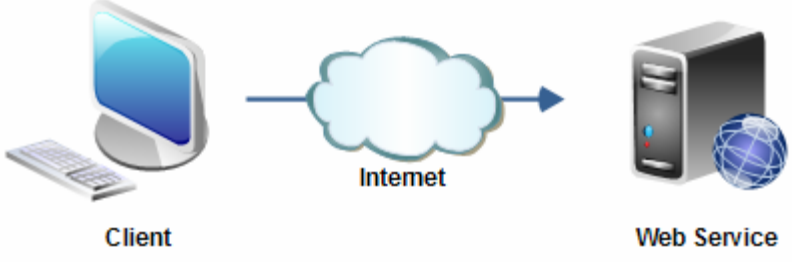
Les serveurs d'application

- **Hébergent les composants** et gèrent leur cycle de vie
- Fournissent d'**autres services**
  - Lien avec serveur web
  - Lien avec SGBD
  - Administration des composants
- Généralement **spécifiques à une technologie**
  - Java EE: JBOSS, Apache TomEE, Oracle GlassFish
  - .Net: intégré au framework .Net
  - PHP: Zend Server
  - Python: Zope

Services web

C'est quoi ?

- Un service Web est une **application logicielle** accessible à partir du **web**. Il utilise les **protocoles internet** pour communiquer et utilise un **langage standard** pour décrire son interface



.center]

- Quelle différence avec un site web ?
  - La **présentation** des informations est inutile (HTML)

La définition du W3C

> *Un service web est un **système logiciel** identifié par **un URI**, dont les interfaces publiques et les « bindings » sont définies et décrites en **XML**. Sa définition peut être **découverte** [dynamiquement] par d'autres **systèmes logiciels**. Ces autres systèmes peuvent ensuite interagir avec le service web d'une façon décrite par sa définition, en utilisant des **messages XML** transportés par des *protocoles Internet*

*Le W3C met en avant XML comme langage de description (à différents niveaux). Mais ce n'est pas le seul moyen (standard) de communication...

Exemples...

- Nombreuses **API** disponibles sur le web
  - Google: https://developers.google.com/apis-explorer (REST)
  - Twitter: https://dev.twitter.com/ (REST)
  - Facebook: https://developers.facebook.com/ (REST)
  - Paypal: https://developer.paypal.com (SOAP)
  - Viamichelin: http://dev.viamichelin.fr/presentation-soap.html (SOAP)
- Accès à ces API via les protocoles de service web (SOAP, REST, etc.)

ou

- Via des kit de développement (SDK) spécifiques (JavaScript, Java, etc.)
  - Les SDK ne font que simplifier l'accès au service web

Les formats d'échange de données web

XML

- **Extensible Markup Language**
- Vu dans les cours précédents (*Remi Lehn*)
- Langage de **description** de données
- On peut **créer de nouveaux langages** via un Schéma XML
- On peut **transformer** un document XML vers d'autres formats (XML ou non) via XSLT
- La base de nombreux services web
- **Attention:** format particulièrement *verbeux*

JSON

- Format de description de données textuel
  - Existe en version binaire (BSON)
- Dérivé de la **notation objet de JavaScript**
- Ne contient que 2 types de structures
  - **Objet** = ensemble non ordonnée de paires `"clé" : valeur`
    - Ex: `{ "nom": "Polytech", "nbEtudiants: 500" }`
  - **Tableau** = collection ordonnée de valeurs
    - Ex: `[ 1, 2, 3, 4, 5]`
- Une **valeur** peut être:
  - Un objet
  - Un tableau
  - Un type simple (chaine, nombre, booléen, null)

XML vs. JSON: exemple

XML

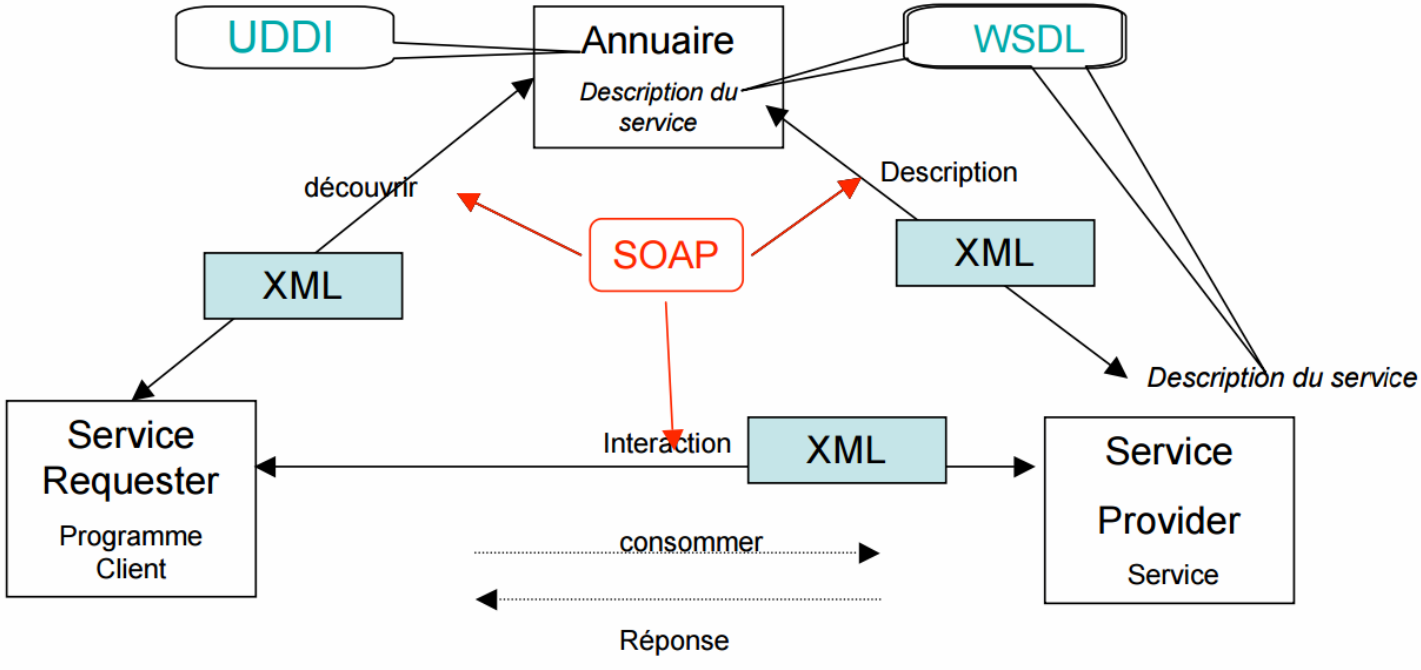
```
<product>
     <id>15</id>
     <name>Widgets</name>
     <description>These widgets are
the finest widgets ever made by
anyone.</description>
     <options type="color">
          <item>Purple</item>
          <item>Green</item>
          <item>Orange</item>
     </options>
</product>
```

JSON

```
"product" : {
     "id" : 15,
     "name" : "Widgets",
     "description" : "These widgets
are the finest widgets ever made by
anyone.",
     "options" : [
          {
               "type" : "color",
               "items" : [
                    "Purple",
                    "Green",
                    "Orange"
               ]
          }
     ]
}
```

Architectures (web) orientées service

Service web (SOAP): architecture



Les 3 briques d'un service web

- **Annuaire** (Service Registry)
  - Annuaire des services publiés par les providers (UDDI)
  - Géré sur un serveur niveau application, entreprise ou mondial
- **Service Provider**
  - Application s'exécutant sur un serveur et comportant un module logiciel accessible en XML
- **Service Requester**
  - Application cliente se liant à un service et invoquant ses fonctions par des messages XML (REST, XML-RPC,SOAP)

SOAP

- Protocole de **communication** de messages
- N'est **pas lié** à un protocole de **transport** particulier (mais HTTP est populaire)
- Un message SOAP est composé de
  - Une **déclaration** XML (optionnelle), suivie de
  - Une **enveloppe** SOAP (l'élément racine) qui est composée de:
    - Un Entête SOAP (optionnel)
    - Un Corps SOAP : dont le contenu dépend de l'application
      - Peut contenir des messages d'erreur et pièces jointes
- Un dialogue SOAP contient un message de **requête** et un message de **réponse**

SOAP

Exemple de dialogue simpliste

Requête: appel d'une méthode qui double la valeur d'un entier

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:doublerUnEntierReq
         xmlns:ns1="urn:MonServiceSOAP">
            <param1 xsi:type="xsd:int">1024</param1>
        </ns1:doublerUnEntierReq>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

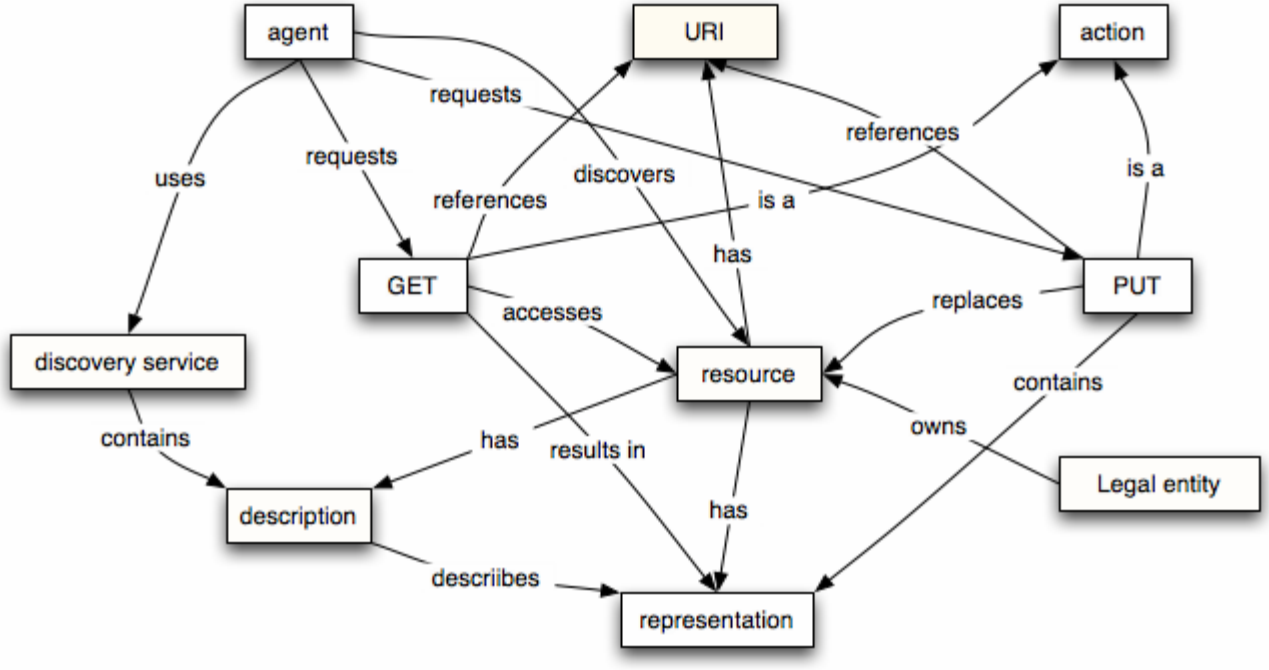
SOAP (exemple dialogue)

Exemple de dialogue simpliste

Réponse: la valeur de l'entier doublé...

```
<?xml version="1.0" encoding="UTF-8" ?>
  <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:doublerUnEntierRep
         xmlns:ns1="urn:MonServiceSOAP"
         SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <return xsi:type="xsd:int">2048</return>
        </ns1:doublerUnEntierResp>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Architectures orientées ressources



REST ? (1/2)

- REST: **REpresentational State Transfer**
- **Style d'architecture** != protocole de communication (SOAP)
- Décrit en 2000 par Roy Thomas Fielding dans sa thèse, chap 5, "Architectural Styles and the Design of Network-based Software Architectures"
  - 1 des principaux acteur de la spécification de HTTP
  - Membre fondateur de la fondation Apache
  - Développeur du serveur Web Apache
- Style d'architecture **inspiré du Web**
- Architecture **orientée ressource**

REST ? (2/2)

- Utilisé pour développer des **Services Web**
- Dans la littérature
  - Architectures orientées **ressources** (ROA)
  - Architectures orientées **données** (DOA)
- Quand une application respecte ces principes: **RESTFul**
- Les services web REST sont **sans états** (Stateless)
  - Pas de mémoire des requêtes antérieures
  - Chaque requête envoyée doit contenir toutes les informations nécessaires au traitement

Les concepts de REST

- **Ressources** (Identifiant)
  - Identifiée par une URI
  - Exemple (fictif): http://www.achats.fr/livre/SF/Harry-Potter
- **Méthodes** (Verbes)
  - Action à effectuer sur la ressource
  - Méthodes HTTP: `GET`, `POST`, `PUT` et `DELETE`
- **Représentation** (Vue de la ressource ou de son état)
  - Informations échangées avec le service
  - Texte, XML, JSON, ...

REST: Ressource

- Tout ce qui est **identifiable / manipulable** dans le système
  - Document, Image, Personne, Le montant du compte d'un client, etc.
- **Identifié par un lien** (URI)
- Une ressource **peut avoir plusieurs URI**
- Une URI identifie **une seule ressource** (ou un seul groupe de ressources)
- Construite de façon **hiérarchique**
- La représentation d'une ressource **peut évoluer avec le temps**
  - Lié au temps: ex. Dernier article
  - Modification structure: ex. Ajout d'un champ

REST: Ressource

- Structure classique
  - Structure **hiérarchique**
  - Construction classique
    - `http://domaine.com/<plus général>/../<plus spécifique>`
- Exemples d'URIs
  - /musique/rock
  - /musique/rock/AC-DC/
  - /musique/rock/AC-DC/année
  - /musique/rock/AC-DC/back_in_black
  - /musique/rock/AC-DC/année/5
  - /musique/classique/meilleures_ventes
  - /musique/recherche/foxy_lady

REST: Ressource

- On peut aussi utiliser la **chaine de requête**
  - Préciser la demande de ressource
  - Utiliser et combiner des critères non hierarchiques
- Exemples
  - Uniquement les dix premiers résultats
    - /music/rock?limit=10
  - Trie par ordre ascendant ou descendant sur un champ particulier
    - /music/rock/AC-DC?sort=asc&sortby=year
  - Format (possible aussi via entête `Accept`)
    - /music/classical/best_sellers?format=json

REST: Interface / methodes

- REST fournit une interface uniforme
- Chaque ressource peut subir **4 opérations de base** (CRUD)
  - Create (Créer)
  - Retrieve (Lire)
  - Update (MAJ)
  - Delete (Supprimer)
- REST **s'appuie sur HTTP** pour exprimer les opérations via les méthodes HTTP
  - `POST` (Créer)
  - `GET` (Lire)
  - `PUT` (MAJ complète) + `PATCH` (MAJ partielle)
  - `DELETE` (Supprimer)
- Il n'est pas nécessaire d'implémenter toutes les méthodes pour une ressource

REST: Représentation

- **Format d'échange** des données
  - Pour le client (`GET`)
  - Pour le serveur (`PUT`, `POST` ou `PATCH`)
- Généralement: Texte, JSON, XML, HTML, CSV
- Le format d'entrée (`POST`) et de sortie (`GET`) d'une même ressource peut varier
- On peut spécifier le format via
  - **Entêtes HTTP** (type MIME via `content-type`)
  - **L'URL** de la ressource
    - directement (`/musique/rock/xml` ou `/musique/rock/json`)
    - via la chaine de requête (`/musique/rock?format=xml` ou `/musique/rock?format=json`)

REST: Limites

- **Actions binaires**:
  - Demander si un artiste fait partie d'un groupe ?
  - Demander si un groupe contient un artiste ?
  - Plus dur : lien entre groupe et artiste lors de la création ?
- **Transactions**
  - Décrire un virement bancaire ?
  - Créer une ressource incrémentalement ?
- Les URI doivent-elles être **évidentes ou opaques** ?
  - Donnée: `/serveur/home/~maurice/` ("joli", mnémotechnique)
  - Identifiant: `/homedir/23eab89c/` (résistant au changement)
- **Sécurité**
  - Authentification HTTP simple
  - Utilisation d'HTTPS, cookies, tokens, etc.

Service vs. ressource (1/2)

- Modèle d'interaction :
  - SOAP : **Échanges**
    - Le serveur conserve des données sur la session
    - Les messages ne contiennent que ce qu'ils expriment
  - REST : **Opérations indépendantes**
    - Serveur sans état
    - Les messages doivent embarquer le contexte
- Cible :
  - SOAP : plutôt des **services transactionnels** (ex: réservation de billet)
  - REST : plutôt des échanges de **données / documents** (ex: tweets)

Service vs. ressource (1/2)

- Protocole :
  - SOAP: **subit HTTP**
    - Indépendant du transport donc d'HTTP, mais la large majorité des échanges passe par HTTP
    - Propre modèle de sécurité
    - Propre retour des erreurs
    - Propre stratégie de cache
  - REST: **épouse HTTP**
- Formats :
  - REST: **s'adapte** aux capacités du client (négociation de contenu)
  - WSDL: **définit finement** les formats des données échangées
- Documentation :
  - REST : **pas de norme**, mais possible depuis WSDL 2.0
  - WSDL : **définit finement** (mais verbeusement) les échanges

SOAP vs. REST : conclusion

REST	SOAP
Suppose une communication point à point	Gère les environnements distribués
Ne nécessite qu'un serveur HTTP	Nécessite des outils / middleware
Pas de description des services	WSDL
Ne fonctionne que via HTTP	Différents types de transport possibles
Peu verbeux	Très verbeux
Pas Standard	Standard
Simple, peu de fonctionnalités gérées	Plus complexe, mais plus complet

C'est fini...